# Trabajo Final
# Gestión de la Calidad del Software

Utilizando el trabajo desarrollado en la materia Ingeniería de Software, complete las siguientes tareas.

1. Plan de Remoción de Defectos
   1.1. Utilice una herramienta como *LocMetrics* para contar la cantidad de líneas de código de su proyecto.
   1.2. Elabore un plan de remoción de defectos, estimando la cantidad de defectos que deberían encontrarse en cada etapa.
   1.3. Compare la cantidad de defectos estimada con la cantidad de defectos que encontró al finalizar la materia Ingeniería de Software.
2. Revisión de los requerimientos.
   2.1. ¿Faltan requerimientos?
   2.2. ¿Sobran requerimientos?
   2.3. ¿Está correctamente escritos los requerimientos?
   2.4. Elabore los reportes de revisión necesarios.
   2.5. ¿En ésta etapa, se encontró la cantidad de defectos esperada de acuerdo a su plan de remoción de defectos?
3. Revisión del diseño.
   3.1. Revise todos los documentos de diseño elaborando los reportes de revisión.
   3.2. ¿En ésta etapa, se encontró la cantidad de defectos esperada de acuerdo a su plan de remoción de defectos?
4. Revisión de código.
   4.1. Elija una herramienta de revisión de código fuente y utilicela para revisar el código fuente de su proyecto. (por ejemplo, Review Board).
   4.2. ¿Cuántos defectos encontraron? ¿Fueron encontrados todos los defectos que esperaba encontrar?
5. Análisis estático de código.
   5.1. Ejecute herramientas de análisis estático de código sobre su proyecto de Ingeniería de Software, cómo mínimo debe correr CheckStyle, PMD, PMD CPD y FindBugs.
   5.2. Analice los resultados, determine cuáles de las advertencias y errores encontrados por las herramientas se deben corregir y cuáles no. Para tomar ésta decisión debe contemplar la criticidad del error, el impacto que produce en el producto, el costo que tiene corregirlo y el beneficio que produce. Documente todas las decisiones. Analice nuevamente el código luego de cada corrección (Ver punto opcional Integración Continua).
6. Pruebas Unitarias
   6.1. ¿Qué cantidad de defectos espera encontrar en ésta etapa?
   6.2. Configure una herramienta que mida el porcentaje de código fuente cubierto por pruebas unitarias, por ejemplo JaCoCo (Java Code Coverage). ¿Cuál es su porcentaje de cobertura?
   6.3. Analice la diferencia entre los conceptos "*line coverage*", "*instruction coverage*" y "*branch coverage*". ¿Cuál considera más útil como métrica para un proyecto de software?
   6.4. ¿Qué valor de "*branch coverage*" tiene su proyecto? Considerando métricas como la complejidad ciclomática (McCabe) determine la cantidad de pruebas unitarias que debería realizar para cubrir los caminos independientes en su proyecto (*branch coverage*). ¿Cuántos casos de prueba adicionales necesitaría?

6.5. Durante el proceso de implementación de las pruebas unitarias necesarias para ampliar el valor de cobertura, contabilice la cantidad de defectos que va encontrando en su producto.

6.6. ¿Qué porcentaje de "*branch coverage*" es deseable para un proyecto de software? Justifique.

7. Pruebas de Sistema.

7.1. Determinar que los casos de prueba de sistema planteados en su trabajo final de Ingeniería de Software son suficientes para cubrir toda la funcionalidad o si se necesitan agregar más escenarios. En este último caso, especifique todos los nuevos escenarios.

7.2. Automatizar al menos el 30% de los casos de prueba de sistema. Mientras mayor sea el porcentaje automatizado mejor. Para ésto, utilice herramientas como JBehave y/o Selenium.

7.3. ¿En ésta etapa, se encontró la cantidad de defectos esperada de acuerdo a su plan de remoción de defectos?

8. Considerando que el valor de 1 (hora) de trabajo tiene un costo de $ 100, y que su producto ya fue entregado al finalizar la materia Ingeniería de Software, es decir, el producto ya fue entregado al cliente, ¿qué cantidad de dinero podría haber ahorrado si hubiera aplicado estas técnicas para encontrar y remover defectos durante el desarrollo del proyecto, en lugar de hacerlo después de entregado?

9. Integración Continua.
   Instale un servidor de integración continua como Jenkins, TeamCity, TravisCI, etc. y establezca las configuraciones necesarias para lanzar un nuevo "*build*" con cada commit. Cada "build" debe incluir las siguientes etapas:
   a. Compilación del código fuente. Haga que el "*build*" falle cuando existan errores de compilación.
   b. Correr CheckStyle sobre el código fuente. Haga que el "*build*" falle cuando la cantidad de errores supere la cantidad que había en el "*build*" anterior.
   c. Correr PMD sobre el código fuente. Haga que el "*build*" falle cuando la cantidad de errores supere la cantidad que había en el "*build*" anterior.
   d. Correr PMD CPD sobre el código fuente. Haga que el "*build*" falle cuando la cantidad de errores supere la cantidad que había en el "*build*" anterior.
   e. Correr FindBugs sobre el código fuente. Haga que el "*build*" falle cuando la cantidad de errores supere la cantidad que había en el "*build*" anterior.
   f. Correr las pruebas unitarias. Calculando la cobertura de código. Haga que el build falle cuando alguna prueba no pase o el nivel de cobertura de código baje.
   g. Correr las pruebas de sistema. Haga que el "*build*" falle cuando alguna prueba no pase.

Elaboren un informe a entregar en formato PDF mostrando todo el trabajo realizado y detallando todo lo que crea necesario.

Elaborar una presentación de 15/20 minutos de duración más 5 minutos de preguntas para presentar ante el curso el trabajo realizado.

Enviar por mail un archivo ZIP con el informe, la presentación y todos los elementos que considere necesarios para msotrar su trabajo.